

A Simple Guide to React

Viraj Shoor

A Simple Guide to React

First Edition

By Viraj Shoor

Introduction

React is a popular JavaScript library for building user interfaces, especially for single-page applications. It allows developers to create large web applications that can update and render efficiently in response to data changes. React is maintained by Facebook and a community of individual developers and companies. This guide will introduce you to the fundamentals of React in simple, layman's terms.

Props

Props (short for properties) are inputs to a React component. They are used to pass data from one component to another. Props are read-only, meaning they cannot be modified by the component that receives them.

Example of Props:

```
import React from 'react';

const Greeting = ({ name }) => {
  return <h1>Hello, {name}!</h1>;
};

export default Greeting;
```

In this example, the `Greeting` component receives a `name` prop and uses it to display a greeting message.

State

State is a built-in object that allows components to create and manage their own data. State is local to the component and can be changed over time. When the state of a component changes, the component re-renders.

Example of State:

```
import React, { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click
    </div>
  );
};

export default Counter;
```

Here, `useState` initializes the `count` state variable with 0, and `setCount` updates the `count` state.

Lifecycle Methods

Lifecycle methods are special methods in class components that allow you to run code at specific points in a component's life. For example, when a component is mounted (added to the DOM), updated (data changes), or unmounted (removed from the DOM).

Example of Lifecycle Methods:

```
import React, { Component } from 'react';

class Timer extends Component {
  state = { seconds: 0 };
}
```

```
componentDidMount() {  
  this.interval = setInterval(() => this.setState({ seconds:  
this.state.seconds + 1 })), 1000);  
}  
  
componentWillUnmount() {  
  clearInterval(this.interval);  
}  
  
render() {  
  return <div>Timer: {this.state.seconds} seconds</div>;  
}  
}  
  
export default Timer;
```

In this example, `componentDidMount` sets up a timer when the component mounts, and `componentWillUnmount` clears the timer when the component unmounts.

Context

Context provides a way to pass data through the component tree without having to pass props down manually at every level. It is designed to share data that can be considered global for a tree of React components.

Example of Context:

```
import React, { createContext, useContext } from 'react';  
  
const ThemeContext = createContext('light');  
  
const ThemedComponent = () => {  
  const theme = useContext(ThemeContext);  
  return <div>The current theme is {theme}</div>;  
};  
  
const App = () => {  
  return (  
    <ThemeContext.Provider value="dark">  
      <ThemedComponent />  
    </ThemeContext.Provider>  
  );  
};
```

```
);  
};  
  
export default App;
```

In this example, `ThemeContext` is created and used to provide and consume the current theme value.

Routing

React Router is a standard library for routing in React applications. It enables navigation among views or pages of a React application.

Example of Routing:

```
import React from 'react';  
import { BrowserRouter as Router, Route, Switch, Link } from  
  'react-router-dom';  
  
const Home = () => <h2>Home</h2>;  
const About = () => <h2>About</h2>;  
  
const App = () => {  
  return (  
    <Router>  
      <div>  
        <nav>  
          <Link to="/">Home</Link>  
          <Link to="/about">About</Link>  
        </nav>  
        <Switch>  
          <Route path="/" exact component={Home} />  
          <Route path="/about" component={About} />  
        </Switch>  
      </div>  
    </Router>  
  );  
};  
  
export default App;
```

In this example, `Router` sets up navigation, `Link` creates navigable links, and `Route` defines the mapping between URL paths and components.

Forms

Handling forms in React typically involves managing state to keep track of form inputs.

Example of Forms:

```
import React, { useState } from 'react';

const MyForm = () => {
  const [name, setName] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Form submitted: ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={(e) =>
setName(e.target.value)} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
};

export default MyForm;
```

In this example, `useState` manages the `name` input value, and `handleSubmit` handles form submission.

Event Handling

React uses synthetic events to handle user interactions in a consistent manner across different browsers.

Example of Event Handling:

```
import React from 'react';

const Button = () => {
  const handleClick = () => {
    alert('Button clicked!');
  };

  return <button onClick={handleClick}>Click me</button>;
};

export default Button;
```

In this example, the `handleClick` function is called when the button is clicked.

Conclusion

These fundamental concepts—props, state, lifecycle methods, context, routing, forms, and event handling—are essential for building React applications. By understanding and utilizing these concepts, you can create dynamic, interactive, and maintainable user interfaces.